军棋AI报告

吴章昊（MW）

2016 年 11 月 27 日

Contents

Chapter 1

概述

1.1 我的AI思路

我的AI思路比较简单，就是用一些办法把所有的可能走法遍历一遍在每一种走法中再向下搜两到三层（交替着我和对手走子），到最底层以后返回一个估价，然后分别在每一层中取出一个最利于这一层走子者的分值（即自己的这层取最大值，对手的层取取负的最大值）返回上一层，最终返回到第一层后，比较出所有走法里分数最高的一项，使用这个走法。其中，我的估价函数是通过将临时棋盘扫描一遍，计算场上的子力得出的，而各个棋子的估值是经过一开始的猜测，然后再测试中不断修改得到的。这是最开始的写法，即暴力模拟，这样的写法能够搜索到三层，之后向里面加入了一些剪枝，迭代加深和其他优化使搜索层数提高到了四层，偶尔能够达到五层。

Chapter 2

艰难起步

2.1 背景

听说要做AI的时候，我的内心是崩溃的，作为一个零基础组的同学，连一个OJ题目都没法做好，怎么说的上做AI 呢...从大作业的布置开始，我曾无数次的打开Sample AI又关掉，虽然看懂了其中各个部分的作用，但是依然对于如何实现一个AI毫无想法。直到有一天，计科导的习题课下课之后，在致远楼的电梯上偶遇mercy 助教...

2.2 起步

当时mercy助教给了我一个网址——象棋巫师，mercy助教的这一推荐帮了大忙，进入这个网址之后我看到的是非常齐全的各种对于AI 制作的文章，这些文章基本都是网站作者自己从外文翻译而来的。于是，我开始一边学习这个网站上的内容，一边开始做我自己的AI。（虽然不知多少天，我都是盯着一篇篇译文，一脸茫然，这里我需要特别提到的是，在我满脑子都是那个网站上各种奇技淫巧，多亏了在一节计科导习题课上（为什么又是计科导习题课or2）一位薛震东助教，和我们说，“你们零基础的就先写个暴力模拟的呀”。然后我就找的了方向）

2.3 起名

起名字是个很重要的环节，而不知道为什么（大概是思维江化了）当时我的脑子里就闪过了一个念头，于是就把它命名为Naive AI v0.1

2.4 动起来

做AI的首要问题就是如何让棋子动起来，以及如何判断这一步移动是否是一步好棋。然而对于这两个方面，我都不知道如何是好。于是我想着，不如先让它能动再说吧。于是，我开始想办法让它启动。就动起来而言，当时我遇到了下面的一些问题。

- **如何走一步，如何走多步？**

  对于这问题，我的想法是对于每一个棋子用两层for循环，外层的for循环遍历该棋子的八个方向，而内层for循环遍历从1到这个方向可以走的最多步数，这样就可以将所有的情况模拟一遍。不过，这里面还有下面的一些细节需要处理。

- **过河**

  如何判断由于棋盘中间的铁道是间隔为2的，不能够直接像上面所述的一个一个的遍历，我想到的方法是当向一个方向走一步后用exist函数判断这一步之后的位置是否在棋盘上，若不在棋盘上就向同方向再走一步，这样就可以保证棋子能够顺利的过河

- **最大步数**

  如何判断最大步数这是一个蛮难办的问题，因为就直行而言用上面过河的办法判断是否出了棋盘边界还是可行的，但是对于一些位置来说能否斜着移动，斜着向哪个方向移动，以及如何区分铁路和那些普通的道路，就没有那么好解决了。为了解决这个问题，我打了一张17*5*8 的表ables，即棋盘上每个位置的八个方向各至多可以走多少步，如对于（1，0）这个位置，在ables中记为ables[1][0][]={ 0, 0, 14, 1, 4, 0, 1, 0 }自左侧开始逆时针方向标出各个方位最多可走的步

数，从而每次搜索的时候，只需要访问这个数组就可以知道各个方向可以走的最大步数。这样通过知道每个方向可以走的最大步数，就可以不必担心直接从铁路上飞到普通的道路上去了。（打印这张表是一件非常费时费力的工作，当时希望通过一个程序输出的，但是由于军棋棋盘的规律性并不是特别强，我以为通过程序输出会非常麻烦，于是我便花了一早上的时间纯粹的用手动输入的方式完成了这张表）

- **吃子、对碰**

  这方面的内容在我的AI里就没有什么想法可言了，我通过调用一个函数来判断某个步是否可以走，是否有吃子、对碰等行为，如果有就停止继续向这个方向走，即从内层的for循环中break出来，防止棋子的飞跃。而对于这个函数的写法，在这个Naive的AI里就非常暴力了，直接通过各种各样的特判，返回走这一步以后新位置剩下的棋子。而如何判断这个对方的子是否在行营里的方法和之前最大步数的方法非常相似，我打了一张17*5的表ymap,将是行营的点标记出来，从而通过判断吃子和碰子行为是否发生在行营里来判断走步是否合法。

- **如何遍历自己和对方可以移动的所有棋子**

  这里我先建了一个结构体dirk，其中包含坐标x，y以及生死状态live三个int型数据。之后开一个2*12*3的dirk类型的数组direction，记录每一个棋子的位置和生死情况，其中2代表己方还是对方，12是各方所拥有的棋子种类数，3即各个棋子的个数都不大于3，于是direction数组表示为direction[color][item][the ith item]，这样在遍历棋子的时候就可以不需要对整个棋盘进行扫描，然而相对的，这使得需要维护的数组增加，增大了代码出错的可能性，由于这个数组的更新错误，可能造成移动别人的棋子，移动空格，等等各种非法移动，这些错误便是Naive AI调试时花费我时间最多的地方。

- **工兵如何移动**

  对于工兵奇异的飞行方式，在我刚开始的时候是没有办法的，我只能将它与普通的棋子合并到一起，这样它能够像普通棋子一样移动，但还不会飞行

2.5 估价

在做AI之初，我遇到的问题非常多，其中之一便是如何估价。既然通过前面的方法，我可以尝试除工兵的飞行之外的所有的行棋着法，接下来就是如何判断那一步更加优越，从而选择这一步。

在我的Naive AI中我的处理办法是，在程序中定义两个全局数组变量valuemy[12]和valueop[12]，其中对每一个编号对应的棋子进行赋值，例如将司令代表的0号valuemy[0]赋值为7.8e3，排长对应的7号valuemy[7]赋值为30.0等等，每一次需要估价的时候就对整个临时棋盘进行一次扫描，遇见自己的棋子就在价值val中加上valuemy的值，遇到对手的棋子就减去valueop中对应的值。这样就可以，对每一种棋盘进行估价了，虽然这个方法看似是静态的，无法看到移动后是否会受到危险，但是由于决策时，AI可以考虑4-5层的情况，这样就可以弥补估价的不足。而在之后的改动中我还进一步的对这个估价函数做了一些调整，我将在Chapter 2中详细说明。

2.6 多层搜索

完成了棋子的移动和估价，接下来的问题便是，如何才能高效率的进行多层搜索。这里我需要感谢象棋巫师为我提供的思路，在象棋巫师中有一个被称为最大最小值搜索的方法，这个方法是指，在搜索自己层的时候，程序的目标便是获得估价最高的走法，而在对手层的时候程序的目标即为获得估价最低的走法，也就是说只需要写一个移动棋子的函数negamax，在每尝试移动一个棋子之后递归调用negamax这个函数，就将更新后的临时棋盘传入，只需要更改direction的第一个下标和程序的目标即可（取最大值还是最小值），而我在这个方法的基础上加了一点小技巧，将每次递归调用后的返回值取负，这样就可以使每次调用函数时的目标相同（取最大值）。一层层的向下搜索后当达到了预先设定的搜索层数就对临时棋盘调用估价函数，并返回结果，这样就可以实现最优招发的选取了。

2.7 棋盘更新

对棋盘的更新也是一件异常麻烦的事情，在我的Naive AI中有临时棋盘tmap、位置数组direction等数组需要维护，在每一次生成着法时，程序需要对两个棋盘进行更改并记录它们的初始状态，在完成了递归调用得到返回值，与最大值比较之后，就需要将tmap和direction恢复到之前的状态来取消着法。这样才能保证每一次尝试走法时不会影响起始的局面。而每一次获得message后对tmap和direction的更新，我直接将它放在了change函数里，这样既可以保证棋盘更新的及时性，同时可以防止在决策时更新，从而节约决策时间。这个棋盘的维护非常重要，有非常容易出错，所以在起始时我的Naive AI常出现非法的移动。

2.8 小结

经过了上述的这些方法，我的AI终于实现了能够暴力模拟搜索三层得出最优解。而Naive AI的版本号也提升到了2.0。这期间还有一些小插曲，对于函数的递归调用之后的返回的最大值，一开始被我定义成了全局变量，也就是说在对某一层层的最大值进行更改时会影响到上一层的最大值，因此我花了好长时间查不出为什么我三层的程序会打不过只搜索了一层的，直到有一天的“科学思想背后的小故事”使我突然发现了错误，然后改正了它XD。接下来让我来讲讲，我对于Naive AI的进一步改进。

Chapter 3

改进

3.1 alpha-beta剪枝

Naive AI中剪枝的加入还需要感谢李怿恺同学的指点，开始时我并不能看懂象棋巫师上关于这个剪枝的描述。在一次午餐的时候，李怿恺同学向我讲解了alpha-beta剪枝的原理，我才真正明白了如何使用它。alpha-beta剪枝就是在之前所提及的搜索方法的基础上，在对手层上每得到一个返回值就将它取负后得到emax与之前的第一层的临时的最大值进行比较，如果前者较小，那么说明如果走了前一层尝试的这一步，那么对手走子后局面的估价一定不大于这个emax，所以前一层尝试的对应走法一定不是最优的，那么就可以直接剪掉这个走法之后的搜索，直接return到上一层，尝试下一种走法，这样就可以减少搜索的节点数，使搜索速度加快，从而使AI有可能搜索到4层。

3.2 迭代加深与卡时

Naive AI中迭代加深的加入，也是受了许多dalao的指点，其中包括冯思远、刘啸远等。同样的，虽然这是一个在象棋巫师上出现了的方法，但是我并不能理解其中的奥妙，在我当时看来，似乎迭代加深会造成进行许多无用的计算浪费时间。之后经他们的指点，我理解了，层数每加深一层，搜索的节点数都是指数级增长的，所以前面层数较少时的搜索是可以

作为无穷小量忽略的。所以我放心的在程序里加入了迭代加深。当然加入这个技术是另有目的的，单纯的加入迭代加深，而固定搜索的层数毫无用处，但是如果加上卡时就可以发挥大作用啦。由于不同的局面AI搜索的节点数并不相同，有时候四层的搜索会超出一秒的时间限制，而有时又可以搜索到五层，所以加入迭代加深之后，程序便可以根据剩余的时间，来判断是否继续加大搜索的层数。如此一来，就可以充分的运用所限制的决策时间。

3.3 着法排序

加入迭代加深和卡时之后，我发现我并没有完全的运用迭代加深的好处，因为在我之前的程序中，在迭代加深的过程中，如果继续搜索下一层，那么就直接舍弃了之前的搜索结果。这样的做法确实是一个非常大的浪费。之后我希望能够通过前一次的搜索，根据着法的优劣进行排序，这样就可以在下一次搜索中先对较优的着法进行搜索，配合上alpha-beta剪枝就又可以缩短更多的时间。不过，由于时间的关系，再加上编程能力的限制，Naive AI目前只能将上一次的最优解法优先搜索（我认为这样的操作就足以缩短很多的时间，排序后的搜索可能与这个方法差异不大）。

3.4 动态估价

对于动态估价，我并没有一个非常清晰地思路。在Naive AI中，我只加入了工兵和炸弹的动态估值，其中工兵的动态调整是根据各方工兵的剩余个数进行调整的，通过valuemy[8] = (3 - countgong) * 7e2 + 5e2这样一条算式来计算工兵的分值，而炸弹的分值是根据对方剩余的司令、军长和师长的个数计算出来的。这样的估价以及其中的各个值的大小并没有非常严密的理论依据，这些参数都是在评测的时候一点点手动修改得到的，虽然似乎使Naive AI变强了一些，但是我相信一定还有更完美的估价，如果以后有机会用退火的方法跑一跑这个程序也许能得到一个更好的结果。

3.5 初始布阵

和动态估价相对的当然还有初始布阵，这个和估价的调整一样，都非常的玄学，只是通过评测时的尝试，逐渐修改的，到现在我依然觉得目前的布阵还是不令人满意。

3.6 小结

经历了这么多的优化和调整，Naive AI终于变得不那么naive了，有时候甚至能够料到一眼看不出来的走法，但是它还是非常遗憾的只能停留在四层的深度，尽管每加入一个优化或是改进，Naive AI搜索的时间都在减少，但它毕竟还是太过于naive，难以再向上提升。如果有更多的时间，我也许会再尝试加入象棋巫师里提及的置换表、历史表之类的黑科技，把我的棋盘编号做一个优化等等，提升Naive AI。

Chapter 4

总结

终于到了总结的环节，这是我第一次用latex写东西，排版异常的诡异，请见谅呀。刚开始，得知这个AI作业的时候，我说：“这磨灭了我编程的兴趣。”现在我要郑重的撤回这句话。这个AI作业真的让我体会到了编程的乐趣，每次看到AI的提升对于我来说都是极大的欢喜。这样的经历，大概是很难忘记的了。虽然我的AI中还留着许多的遗憾，但我还是感受到了前所未有的满足感。在这里我要真心地感谢许多人。

- **计科导的助教们（or2）：周宇皓、mercy、薛震东...**

  他们将我引向了正确的道路，使我可以着手写AI。

- **教导我的dalao们:李怿恺、冯思远、刘啸远、闫鸿宇...**

  他们教会了我许多的技术，让我的AI得以成长。

- **以及助教组的各位助教们**

  各位助教们，谢谢你们的默默付出，这是我第一次发现如此关心“同学”的“老师”，每一次服务器傲娇的时候，都会第一时间被修复，每次同学有问题的时候，都会迅速地得到的解答，你们的付出甚至推翻了我对于大学的认识，这里不仅仅有学术，这里还有温暖的人情和传承。谢谢!!

此致
敬礼